Red Hat build of Quarkus 3.2

Configure data sources

Last Updated: 2024-04-04

Abstract

Define JDBC and Reactive driver data sources in Red Hat build of Quarkus using a unified configuration model.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

CHAPTER 1. CONFIGURE DATA SOURCES

Use a unified configuration model to define data sources for Java Database Connectivity (JDBC) and Reactive drivers in Quarkus.

Applications use datasources to access relational databases. Quarkus provides a unified configuration model to define datasources for Java Database Connectivity (JDBC) and Reactive database drivers.

Quarkus uses Agroal and Vert.x to provide high-performance, scalable data source connection pooling for JDBC and reactive drivers. The **jdbc-*** and **reactive-*** extensions provide build time optimizations and integrate configured data sources with Quarkus features like security, health checks, and metrics.

For more information about consuming and using a reactive datasource, see the Quarkus Reactive SQL clients guide.

Additionally, refer to the Quarkus Hibernate ORM guide for information on consuming and using a JDBC datasource.

1.1. GET STARTED WITH CONFIGURING DATASOURCES IN QUARKUS

For users familiar with the fundamentals, this section provides an overview and code samples to set up data sources quickly.

For more advanced configuration with examples, see References.

1.1.1. Zero-config setup in development mode

Quarkus simplifies database configuration by offering the Dev Services feature, enabling zero-config database setup for testing or running in development (dev) mode. In dev mode, the suggested approach is to use DevServices and let Quarkus handle the database for you, whereas for production mode, you provide explicit database configuration details pointing to a database managed outside of Quarkus.

To use Dev Services, add the appropriate driver extension, such as **jdbc-postgresql**, for your desired database type to the **pom.xml** file. In dev mode, if you do not provide any explicit database connection details, Quarkus automatically handles the database setup and provides the wiring between the application and the database.

If you provide user credentials, the underlying database will be configured to use them. This is useful if you want to connect to the database with an external tool.

To use this feature, ensure a Docker or Podman container runtime is installed, depending on the database type. Certain databases, such as H2, operate in in-memory mode and do not require a container runtime.

TIP

Prefix the actual connection details for prod mode with **%prod.** to ensure they are not applied in dev mode. For more information, see the Profiles section of the "Configuration reference" guide.

For more information about Dev Services, see Dev Services overview.

For more details and optional configurations, see Dev Services for databases.

1.1.2. Configure a JDBC datasource

1. Add the correct JDBC extension for the database of your choice.
   - **jdbc-db2**
   - **jdbc-derby**
   - **jdbc-h2**
   - **jdbc-mariadb**
   - **jdbc-mssql**
   - **jdbc-mysql**
   - **jdbc-oracle**
   - **jdbc-postgresql**
2. Configure your JDBC datasource:

```
quarkus.datasource.db-kind=postgresql (1)
quarkus.datasource.username=<your username>
quarkus.datasource.password=<your password>

quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/hibernate_orm_test
quarkus.datasource.jdbc.max-size=16
```

(1) This configuration value is only required if there is more than one database extension on the classpath.

If only one viable extension is available, Quarkus assumes this is the correct one. When you add a driver to the test scope, Quarkus automatically includes the specified driver in testing.

1.1.2.1. JDBC connection pool size adjustment

To protect your database from overloading during load peaks, size the pool adequately to throttle the database load. The optimal pool size depends on many factors, such as the number of parallel application users or the nature of the workload.

Be aware that setting the pool size too low might cause some requests to time out while waiting for a connection.

For more information about pool size adjustment properties, see the JDBC configuration reference section.

1.1.3. Configure a reactive datasource

1. Add the correct reactive extension for the database of your choice.
   - **reactive-db2-client**
   - **reactive-mssql-client**
   - **reactive-mysql-client**

- **reactive-oracle-client**
- **reactive-pg-client**

2. Configure your reactive datasource:

```
quarkus.datasource.db-kind=postgresql (1)
quarkus.datasource.username=<your username>
quarkus.datasource.password=<your password>

quarkus.datasource.reactive.url=postgresql:///your_database
quarkus.datasource.reactive.max-size=20
```

(1) This configuration value is only required if there is more than one Reactive driver extension on the classpath.

1.2. CONFIGURE DATASOURCES

The following section describes the configuration for single or multiple datasources. For simplicity, we will reference a single datasource as the default (unnamed) datasource.

1.2.1. Configure a single datasource

A datasource can be either a JDBC datasource, reactive, or both. This depends on the configuration and the selection of project extensions.

1. Define a datasource with the following configuration property, where **db-kind** defines which database platform to connect to, for example, **h2**:

```
quarkus.datasource.db-kind=h2
```

Quarkus deduces the JDBC driver class it needs to use from the specified value of the **db-kind** database platform attribute.

NOTE

This step is required only if your application depends on multiple database drivers. If the application operates with a single driver, this driver is detected automatically.

Quarkus currently includes the following built-in database kinds:

- DB2: **db2**
- Derby: **derby**
- H2: **h2**
- MariaDB: **mariadb**
- Microsoft SQL Server: **mssql**
- MySQL: **mysql**

- Oracle: **oracle**
- PostgreSQL: **postgresql**, **pgsql** or **pg**
- To use a database kind that is not built-in, use **other** and define the JDBC driver explicitly

NOTE

You can use any JDBC driver in a Quarkus app in JVM mode as described in Using other databases. However, using a non-built-in database kind is unlikely to work when compiling your application to a native executable.

For native executable builds, it is recommended to either use the available JDBC Quarkus extensions or contribute a custom extension for your specific driver.

2. Configure the following properties to define credentials:

```
quarkus.datasource.username=<your username>
quarkus.datasource.password=<your password>
```

You can also retrieve the password from Vault by using a credential provider for your datasource.

Until now, the configuration has been the same regardless of whether you are using a JDBC or a reactive driver. When you have defined the database kind and the credentials, the rest depends on what type of driver you are using. It is possible to use JDBC and a reactive driver simultaneously.

1.2.1.1. JDBC datasource

JDBC is the most common database connection pattern, typically needed when used in combination with non-reactive Hibernate ORM.

1. To use a JDBC datasource, start with adding the necessary dependencies:

   a. For use with a built-in JDBC driver, choose and add the Quarkus extension for your relational database driver from the list below:

   - Derby - **jdbc-derby**
   - H2 - **jdbc-h2**

     NOTE

     H2 and Derby databases can be configured to run in "embedded mode"; however, the Derby extension does not support compiling the embedded database engine into native executables.

     Read Testing with in-memory databases for suggestions regarding integration testing.

   - DB2 - **jdbc-db2**
   - MariaDB - **jdbc-mariadb**

- Microsoft SQL Server - **jdbc-mssql**
- MySQL - **jdbc-mysql**
- Oracle - **jdbc-oracle**
- PostgreSQL - **jdbc-postgresql**
- Other JDBC extensions, such as SQLite and its documentation, can be found in the Quarkiverse.
  For example, to add the PostgreSQL driver dependency:

  ```
  ./mvnw quarkus:add-extension -Dextensions="jdbc-postgresql"
  ```

  NOTE

  Using a built-in JDBC driver extension automatically includes the Agroal extension, which is the JDBC connection pool implementation applicable for custom and built-in JDBC drivers. However, for custom drivers, Agroal needs to be added explicitly.

b. For use with a custom JDBC driver, add the **quarkus-agroal** dependency to your project alongside the extension for your relational database driver:

```
./mvnw quarkus:add-extension -Dextensions="agroal"
```

To use a JDBC driver for another database, use a database with no built-in extension or with a different driver.

2. Configure the JDBC connection by defining the JDBC URL property:

```
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/hibernate_orm_test
```

NOTE

Note the **jdbc** prefix in the property name. All the configuration properties specific to JDBC have the **jdbc** prefix. For reactive datasources, the prefix is **reactive**.

For more information about configuring JDBC, see JDBC URL format reference and Quarkus extensions and database drivers reference.

1.2.1.1.1. Custom databases and drivers

If you need to connect to a database for which Quarkus does not provide an extension with the JDBC driver, you can use a custom driver instead. For example, if you are using the OpenTracing JDBC driver in your project.

Without an extension, the driver will work correctly in any Quarkus app running in JVM mode. However, the driver is unlikely to work when compiling your application to a native executable. If you plan to make a native executable, use the existing JDBC Quarkus extensions, or contribute one for your driver.

An example with the OpenTracing driver:

```
quarkus.datasource.jdbc.driver=io.opentracing.contrib.jdbc.TracingDriver
```

An example for defining access to a database with no built-in support in JVM mode:

```
quarkus.datasource.db-kind=other
quarkus.datasource.jdbc.driver=oracle.jdbc.driver.OracleDriver
quarkus.datasource.jdbc.url=jdbc:oracle:thin:@192.168.1.12:1521/ORCL_SVC
quarkus.datasource.username=scott
quarkus.datasource.password=tiger
```

For all the details about the JDBC configuration options and configuring other aspects, such as the connection pool size, refer to the JDBC configuration reference section.

1.2.1.1.2. Consuming the datasource

With Hibernate ORM, the Hibernate layer automatically picks up the datasource and uses it.

For the in-code access to the datasource, obtain it as any other bean as follows:

```
@Inject
AgroalDataSource defaultDataSource;
```

In the above example, the type is **AgroalDataSource**, a **javax.sql.DataSource** subtype. Because of this, you can also use **javax.sql.DataSource** as the injected type.

1.2.1.2. Reactive datasource

Quarkus offers several reactive clients for use with a reactive datasource.

1. Add the corresponding extension to your application:
   - DB2: **quarkus-reactive-db2-client**
   - MariaDB/MySQL: **quarkus-reactive-mysql-client**
   - Microsoft SQL Server: **quarkus-reactive-mssql-client**
   - Oracle: **quarkus-reactive-oracle-client**
   - PostgreSQL: **quarkus-reactive-pg-client**
     The installed extension must be consistent with the **quarkus.datasource.db-kind** you define in your datasource configuration.
2. After adding the driver, configure the connection URL and define a proper size for your connection pool.

   ```
   quarkus.datasource.reactive.url=postgresql:///your_database
   quarkus.datasource.reactive.max-size=20
   ```

1.2.1.2.1. Reactive connection pool size adjustment

To protect your database from overloading during load peaks, size the pool adequately to throttle the database load. The proper size always depends on many factors, such as the number of parallel application users or the nature of the workload.

Be aware that setting the pool size too low might cause some requests to time out while waiting for a connection.

For more information about pool size adjustment properties, see the Reactive datasource configuration reference section.

1.2.1.3. JDBC and reactive datasources simultaneously

When a JDBC extension - along with Agroal - and a reactive datasource extension handling the given database kind are included, they will both be created by default.

- To disable the JDBC datasource explicitly:

  ```
  quarkus.datasource.jdbc=false
  ```

- To disable the reactive datasource explicitly:

  ```
  quarkus.datasource.reactive=false
  ```

TIP

In most cases, the configuration above will be optional as either a JDBC driver or a reactive datasource extension will be present, not both.

1.2.2. Configure multiple datasources

NOTE

The Hibernate ORM extension supports defining persistence units by using configuration properties. For each persistence unit, point to the datasource of your choice.

Defining multiple datasources works like defining a single datasource, with one important change - you have to specify a name (configuration key) for each datasource.

The following example provides three different datasources:

- the default one
- a datasource named **users**
- a datasource named **inventory**

Each with its configuration:

```
quarkus.datasource.db-kind=h2
quarkus.datasource.username=username-default
quarkus.datasource.jdbc.url=jdbc:h2:mem:default
quarkus.datasource.jdbc.max-size=13

quarkus.datasource.users.db-kind=h2
quarkus.datasource.users.username=username1
quarkus.datasource.users.jdbc.url=jdbc:h2:mem:users
quarkus.datasource.users.jdbc.max-size=11
```

```
quarkus.datasource.inventory.db-kind=h2
quarkus.datasource.inventory.username=username2
quarkus.datasource.inventory.jdbc.url=jdbc:h2:mem:inventory
quarkus.datasource.inventory.jdbc.max-size=12
```

Notice there is an extra section in the configuration key. The syntax is as follows: **quarkus.datasource.[optional name.][datasource property]**.

NOTE

Even when only one database extension is installed, named databases need to specify at least one build-time property so that Quarkus can detect them. Generally, this is the **db-kind** property, but you can also specify Dev Services properties to create named datasources according to the Dev Services for Databases guide.

1.2.2.1. Named datasource injection

When using multiple datasources, each **DataSource** also has the **io.quarkus.agroal.DataSource** qualifier with the name of the datasource as the value.

By using the properties mentioned in the previous section to configure three different datasources, inject each one of them as follows:

```
@Inject
AgroalDataSource defaultDataSource;

@Inject
@DataSource("users")
AgroalDataSource usersDataSource;

@Inject
@DataSource("inventory")
AgroalDataSource inventoryDataSource;
```

1.3. DATASOURCE INTEGRATIONS

1.3.1. Datasource health check

If you use the **quarkus-smallrye-health** extension, the **quarkus-agroal** and reactive client extensions automatically add a readiness health check to validate the datasource.

When you access your application's health readiness endpoint, **/q/health/ready** by default, you receive information about the datasource validation status. If you have multiple datasources, all datasources are checked, and if a single datasource validation failure occurs, the status changes to **DOWN**.

This behavior can be disabled by using the **quarkus.datasource.health.enabled** property.

To exclude only a particular datasource from the health check, use:

```
quarkus.datasource."datasource-name".health-exclude=true
```

1.3.2. Datasource metrics

If you are using the **quarkus-micrometer** or **quarkus-smallrye-metrics** extension, **quarkus-agroal** can contribute some datasource-related metrics to the metric registry. This can be activated by setting the **quarkus.datasource.metrics.enabled** property to **true**.

For the exposed metrics to contain any actual values, a metric collection must be enabled internally by the Agroal mechanisms. By default, this metric collection mechanism is enabled for all datasources when a metrics extension is present, and metrics for the Agroal extension are enabled.

To disable metrics for a particular data source, set **quarkus.datasource.jdbc.enable-metrics** to **false**, or apply **quarkus.datasource.<datasource name>.jdbc.enable-metrics** for a named datasource. This disables collecting the metrics and exposing them in the **/q/metrics** endpoint if the mechanism to collect them is disabled.

Conversely, setting **quarkus.datasource.jdbc.enable-metrics** to **true**, or **quarkus.datasource.<datasource name>.jdbc.enable-metrics** for a named datasource explicitly enables metrics collection even if a metrics extension is not in use. This can be useful if you need to access the collected metrics programmatically. They are available after calling **dataSource.getMetrics()** on an injected **AgroalDataSource** instance.

If the metrics collection for this datasource is disabled, all values result in zero.

1.3.3. Narayana transaction manager integration

Integration is automatic if the Narayana JTA extension is also available.

You can override this by setting the **transactions** configuration property:

- **quarkus.datasource.jdbc.transactions** for default unnamed datasource
- **quarkus.datasource.*<datasource-name>*.jdbc.transactions** for named datasource

For more information, see the Configuration reference section below.

To facilitate the storage of transaction logs in a database by using JDBC, see Configuring transaction logs to be stored in a datasource section of the Using transactions in Quarkus guide.

1.3.3.1. Named datasources

When using Dev Services, the default datasource will always be created, but to specify a named datasource, you need to have at least one build time property so Quarkus can detect how to create the datasource.

You will usually specify the **db-kind** property or explicitly enable Dev Services by setting **quarkus.datasource."name".devservices.enabled=true**.

1.3.4. Testing with in-memory databases

Some databases like H2 and Derby are commonly used in the *embedded mode* as a facility to run integration tests quickly.

The recommended approach is to use the real database you intend to use in production, especially when Dev Services provide a zero-config database for testing, and running tests against a container is relatively quick and produces expected results on an actual environment. However, it is also possible to use JVM-powered databases for scenarios when the ability to run simple integration tests is required.

1.3.4.1. Support and limitations

Embedded databases (H2 and Derby) work in JVM mode. For native mode, the following limitations apply:

- Derby cannot be embedded into the application in native mode. However, the Quarkus Derby extension allows native compilation of the Derby JDBC **client**, supporting **remote** connections.
- Embedding H2 within your native image is not recommended. Consider using an alternative approach, for example, using a remote connection to a separate database instead.

1.3.4.2. Run an integration test

1. Add a dependency on the artifacts providing the additional tools that are under the following Maven coordinates:
   - **io.quarkus:quarkus-test-h2** for H2
   - **io.quarkus:quarkus-test-derby** for Derby
     This will allow you to test your application even when it is compiled into a native executable while the database will run as a JVM process.
2. Add the following specific annotation on any class in your integration tests for running integration tests in both JVM or native executables:
   - **@QuarkusTestResource(H2DatabaseTestResource.class)**
   - **@QuarkusTestResource(DerbyDatabaseTestResource.class)**
     This ensures that the test suite starts and terminates the managed database in a separate process as required for test execution.

     H2 example

     ```
     package my.app.integrationtests.db;

     import io.quarkus.test.common.QuarkusTestResource;
     import io.quarkus.test.h2.H2DatabaseTestResource;

     @QuarkusTestResource(H2DatabaseTestResource.class)
     public class TestResources {
     }
     ```
3. Configure the connection to the managed database:

   ```
   quarkus.datasource.db-kind=h2
   quarkus.datasource.jdbc.url=jdbc:h2:tcp://localhost/mem:test
   ```

1.4. REFERENCES

1.4.1. Common datasource configuration reference

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
🔒 **quarkus.datasource.db-kind** The kind of database we will connect to (e.g. h2, postgresql…). Environment variable: **QUARKUS_DATASOURCE_DB_KIND**	string	
🔒 **quarkus.datasource.db-version** The version of the database we will connect to (e.g. '10.0'). CAUTION The version number set here should follow the same numbering scheme as the string returned by **java.sql.DatabaseMetaData#getDatabaseProductVersion()** for your database's JDBC driver. This numbering scheme may be different from the most popular one for your database; for example Microsoft SQL Server 2016 would be version **13**. As a rule, the version set here should be as high as possible, but must be lower than or equal to the version of any database your application will connect to. A high version will allow better performance and using more features (e.g. Hibernate ORM may generate more efficient SQL, avoid workarounds and take advantage of more database features), but if it is higher than the version of the database you want to connect to, it may lead to runtime exceptions (e.g. Hibernate ORM may generate invalid SQL that your database will reject). Some extensions (like the Hibernate ORM extension) will try to check this version against the actual database version on startup, leading to a startup failure when the actual version is lower or simply a warning in case the database cannot be reached. The default for this property is specific to each extension; the Hibernate ORM extension will default to the oldest version it supports. Environment variable: **QUARKUS_DATASOURCE_DB_VERSION**	string	
🔒 **quarkus.datasource.devservices.enabled** If DevServices has been explicitly enabled or disabled. DevServices is generally enabled by default unless an existing configuration is present. When DevServices is enabled, Quarkus will attempt to automatically configure and start a database when running in Dev or Test mode. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_ENABLED**	boolean	

🔒 **quarkus.datasource.devservices.image-name** The container image name for container-based DevServices providers. This has no effect if the provider is not a container-based database, such as H2 or Derby. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_IMAGE_NAME**	string	
🔒 **quarkus.datasource.devservices.port** Optional fixed port the dev service will listen to. If not defined, the port will be chosen randomly. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_PORT**	int	
🔒 **quarkus.datasource.devservices.command** The container start command to use for container-based DevServices providers. This has no effect if the provider is not a container-based database, such as H2 or Derby. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_COMMAND**	string	
🔒 **quarkus.datasource.devservices.db-name** The database name to use if this Dev Service supports overriding it. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_DB_NAME**	string	
🔒 **quarkus.datasource.devservices.username** The username to use if this Dev Service supports overriding it. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_USERNAME**	string	
🔒 **quarkus.datasource.devservices.password** The password to use if this Dev Service supports overriding it. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_PASSWORD**	string	
🔒 **quarkus.datasource.devservices.init-script-path** The path to a SQL script to be loaded from the classpath and applied to the Dev Service database. This has no effect if the provider is not a container-based database, such as H2 or Derby. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_INIT_SCRIPT_PATH**	string	

🔒 **quarkus.datasource.health-exclude** Whether this particular data source should be excluded from the health check if the general health check for data sources is enabled. By default, the health check includes all configured data sources (if it is enabled). Environment variable: **QUARKUS_DATASOURCE_HEALTH_EXCLUDE**	boolean	**false**
🔒 **quarkus.datasource.health.enabled** Whether or not a health check is published in case the smallrye-health extension is present. This is a global setting and is not specific to a datasource. Environment variable: **QUARKUS_DATASOURCE_HEALTH_ENABLED**	boolean	**true**
🔒 **quarkus.datasource.metrics.enabled** Whether or not datasource metrics are published in case a metrics extension is present. This is a global setting and is not specific to a datasource. NOTE This is different from the "jdbc.enable-metrics" property that needs to be set on the JDBC datasource level to enable collection of metrics for that datasource. Environment variable: **QUARKUS_DATASOURCE_METRICS_ENABLED**	boolean	**false**
quarkus.datasource.username The datasource username Environment variable: **QUARKUS_DATASOURCE_USERNAME**	string	
quarkus.datasource.password The datasource password Environment variable: **QUARKUS_DATASOURCE_PASSWORD**	string	
quarkus.datasource.credentials-provider The credentials provider name Environment variable: **QUARKUS_DATASOURCE_CREDENTIALS_PROVIDER**	string	

quarkus.datasource.credentials-provider-name The credentials provider bean name. This is a bean name (as in **@Named**) of a bean that implements **CredentialsProvider**. It is used to select the credentials provider bean when multiple exist. This is unnecessary when there is only one credentials provider available. For Vault, the credentials provider bean name is **vault-credentials-provider**. Environment variable: **QUARKUS_DATASOURCE_CREDENTIALS_PROVIDER_NAME**	string	
🔒 **quarkus.datasource.devservices.container-env** Environment variables that are passed to the container. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_CONTAINER_ENV**	**Map<String,String>**	
🔒 **quarkus.datasource.devservices.container-properties** Generic properties that are passed for additional container configuration. Properties defined here are database-specific and are interpreted specifically in each database dev service implementation. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_CONTAINER_PROPERTIES**	**Map<String,String>**	
🔒 **quarkus.datasource.devservices.properties** Generic properties that are added to the database connection URL. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_PROPERTIES**	**Map<String,String>**	
🔒 **quarkus.datasource.devservices.volumes** The volumes to be mapped to the container. The map key corresponds to the host location; the map value is the container location. If the host location starts with "classpath:", the mapping loads the resource from the classpath with read-only permission. When using a file system location, the volume will be generated with read-write permission, potentially leading to data loss or modification in your file system. This has no effect if the provider is not a container-based database, such as H2 or Derby. Environment variable: **QUARKUS_DATASOURCE_DEVSERVICES_VOLUMES**	**Map<String,String>**	
Additional named datasources	Type	Default

🔒 **quarkus.datasource."datasource-name".db-kind** The kind of database we will connect to (e.g. h2, postgresql…). Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DB_KIND**	string	
🔒 **quarkus.datasource."datasource-name".db-version** The version of the database we will connect to (e.g. '10.0'). CAUTION The version number set here should follow the same numbering scheme as the string returned by **java.sql.DatabaseMetaData#getDatabaseProductVersion()** for your database's JDBC driver. This numbering scheme may be different from the most popular one for your database; for example Microsoft SQL Server 2016 would be version **13**. As a rule, the version set here should be as high as possible, but must be lower than or equal to the version of any database your application will connect to. A high version will allow better performance and using more features (e.g. Hibernate ORM may generate more efficient SQL, avoid workarounds and take advantage of more database features), but if it is higher than the version of the database you want to connect to, it may lead to runtime exceptions (e.g. Hibernate ORM may generate invalid SQL that your database will reject). Some extensions (like the Hibernate ORM extension) will try to check this version against the actual database version on startup, leading to a startup failure when the actual version is lower or simply a warning in case the database cannot be reached. The default for this property is specific to each extension; the Hibernate ORM extension will default to the oldest version it supports. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DB_VERSION**	string	
🔒 **quarkus.datasource."datasource-name".devservices.enabled** If DevServices has been explicitly enabled or disabled. DevServices is generally enabled by default unless an existing configuration is present. When DevServices is enabled, Quarkus will attempt to automatically configure and start a database when running in Dev or Test mode. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_ENABLED**	boolean	

🔒 quarkus.datasource."datasource-name".devservices.image-name The container image name for container-based DevServices providers. This has no effect if the provider is not a container-based database, such as H2 or Derby. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_IMAGE_NAME**	string	
🔒 quarkus.datasource."datasource-name".devservices.container-env Environment variables that are passed to the container. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_CONTAINER_ENV**	**Map<String,String>**	
🔒 quarkus.datasource."datasource-name".devservices.container-properties Generic properties that are passed for additional container configuration. Properties defined here are database-specific and are interpreted specifically in each database dev service implementation. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_CONTAINER_PROPERTIES**	**Map<String,String>**	
🔒 quarkus.datasource."datasource-name".devservices.properties Generic properties that are added to the database connection URL. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_PROPERTIES**	**Map<String,String>**	
🔒 quarkus.datasource."datasource-name".devservices.port Optional fixed port the dev service will listen to. If not defined, the port will be chosen randomly. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_PORT**	int	

🔒 **quarkus.datasource."datasource-name".devservices.command** The container start command to use for container-based DevServices providers. This has no effect if the provider is not a container-based database, such as H2 or Derby. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_COMMAND**	string	
🔒 **quarkus.datasource."datasource-name".devservices.db-name** The database name to use if this Dev Service supports overriding it. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_DB_NAME**	string	
🔒 **quarkus.datasource."datasource-name".devservices.username** The username to use if this Dev Service supports overriding it. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_USERNAME**	string	
🔒 **quarkus.datasource."datasource-name".devservices.password** The password to use if this Dev Service supports overriding it. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_PASSWORD**	string	
🔒 **quarkus.datasource."datasource-name".devservices.init-script-path** The path to a SQL script to be loaded from the classpath and applied to the Dev Service database. This has no effect if the provider is not a container-based database, such as H2 or Derby. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_INIT_SCRIPT_PATH**	string	

🔒 quarkus.datasource."datasource-name".devservices.volumes The volumes to be mapped to the container. The map key corresponds to the host location; the map value is the container location. If the host location starts with "classpath:", the mapping loads the resource from the classpath with read-only permission. When using a file system location, the volume will be generated with read-write permission, potentially leading to data loss or modification in your file system. This has no effect if the provider is not a container-based database, such as H2 or Derby. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__DEVSERVICES_VOLUMES**	**Map<String,String>**	
🔒 quarkus.datasource."datasource-name".health-exclude Whether this particular data source should be excluded from the health check if the general health check for data sources is enabled. By default, the health check includes all configured data sources (if it is enabled). Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__HEALTH_EXCLUDE**	boolean	**false**
quarkus.datasource."datasource-name".username The datasource username Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__USERNAME**	string	
quarkus.datasource."datasource-name".password The datasource password Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__PASSWORD**	string	
quarkus.datasource."datasource-name".credentials-provider The credentials provider name Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__CREDENTIALS_PROVIDER**	string	

quarkus.datasource."datasource-name".credentials-provider-name The credentials provider bean name. This is a bean name (as in **@Named**) of a bean that implements **CredentialsProvider**. It is used to select the credentials provider bean when multiple exist. This is unnecessary when there is only one credentials provider available. For Vault, the credentials provider bean name is **vault-credentials-provider**. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__CREDENTIALS_PROVIDER_NAME**	string	

1.4.2. JDBC configuration reference

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
🔒 quarkus.datasource.jdbc If we create a JDBC datasource for this datasource. Environment variable: **QUARKUS_DATASOURCE_JDBC**	boolean	**true**
🔒 quarkus.datasource.jdbc.driver The datasource driver class name Environment variable: **QUARKUS_DATASOURCE_JDBC_DRIVER**	string	
🔒 quarkus.datasource.jdbc.transactions Whether we want to use regular JDBC transactions, XA, or disable all transactional capabilities. When enabling XA you will need a driver implementing **javax.sql.XADataSource**. Environment variable: **QUARKUS_DATASOURCE_JDBC_TRANSACTIONS**	**enabled**, **xa**, **disabled**	**enabled**
🔒 quarkus.datasource.jdbc.enable-metrics Enable datasource metrics collection. If unspecified, collecting metrics will be enabled by default if a metrics extension is active. Environment variable: **QUARKUS_DATASOURCE_JDBC_ENABLE_METRICS**	boolean	

🔒 **quarkus.datasource.jdbc.tracing** Enable JDBC tracing. Disabled by default. Environment variable: **QUARKUS_DATASOURCE_JDBC_TRACING**	boolean	**false**
🔒 **quarkus.datasource.jdbc.telemetry** Enable OpenTelemetry JDBC instrumentation. Environment variable: **QUARKUS_DATASOURCE_JDBC_TELEMETRY**	boolean	**false**
quarkus.datasource.jdbc.url The datasource URL Environment variable: **QUARKUS_DATASOURCE_JDBC_URL**	string	
quarkus.datasource.jdbc.initial-size The initial size of the pool. Usually you will want to set the initial size to match at least the minimal size, but this is not enforced so to allow for architectures which prefer a lazy initialization of the connections on boot, while being able to sustain a minimal pool size after boot. Environment variable: **QUARKUS_DATASOURCE_JDBC_INITIAL_SIZE**	int	
quarkus.datasource.jdbc.min-size The datasource pool minimum size Environment variable: **QUARKUS_DATASOURCE_JDBC_MIN_SIZE**	int	**0**
quarkus.datasource.jdbc.max-size The datasource pool maximum size Environment variable: **QUARKUS_DATASOURCE_JDBC_MAX_SIZE**	int	**20**
quarkus.datasource.jdbc.background-validation-interval The interval at which we validate idle connections in the background. Set to **0** to disable background validation. Environment variable: **QUARKUS_DATASOURCE_JDBC_BACKGROUND_VALIDATION_INTERVAL**	Duration ?	**2M**

quarkus.datasource.jdbc.foreground-validation-interval Perform foreground validation on connections that have been idle for longer than the specified interval. Environment variable: **QUARKUS_DATASOURCE_JDBC_FOREGROUND_VALIDATION_INTERVAL**	Duration ⓘ	
quarkus.datasource.jdbc.acquisition-timeout The timeout before cancelling the acquisition of a new connection Environment variable: **QUARKUS_DATASOURCE_JDBC_ACQUISITION_TIMEOUT**	Duration ⓘ	**5**
quarkus.datasource.jdbc.leak-detection-interval The interval at which we check for connection leaks. Environment variable: **QUARKUS_DATASOURCE_JDBC_LEAK_DETECTION_INTERVAL**	Duration ⓘ	**This feature is disabled by default.**
quarkus.datasource.jdbc.idle-removal-interval The interval at which we try to remove idle connections. Environment variable: **QUARKUS_DATASOURCE_JDBC_IDLE_REMOVAL_INTERVAL**	Duration ⓘ	**5M**
quarkus.datasource.jdbc.max-lifetime The max lifetime of a connection. Environment variable: **QUARKUS_DATASOURCE_JDBC_MAX_LIFETIME**	Duration ⓘ	**By default, there is no restriction on the lifespan of a connection.**

quarkus.datasource.jdbc.transaction-isolation-level The transaction isolation level. Environment variable: **QUARKUS_DATASOURCE_JDBC_TRANSACTION_ISOLATION_LEVEL**	**undefined**, **none**, **read-uncommitted**, **read-committed**, **repeatable-read**, **serializable**	
quarkus.datasource.jdbc.extended-leak-report Collect and display extra troubleshooting info on leaked connections. Environment variable: **QUARKUS_DATASOURCE_JDBC_EXTENDED_LEAK_REPORT**	boolean	**false**
quarkus.datasource.jdbc.flush-on-close Allows connections to be flushed upon return to the pool. It’s not enabled by default. Environment variable: **QUARKUS_DATASOURCE_JDBC_FLUSH_ON_CLOSE**	boolean	**false**
quarkus.datasource.jdbc.detect-statement-leaks When enabled, Agroal will be able to produce a warning when a connection is returned to the pool without the application having closed all open statements. This is unrelated with tracking of open connections. Disable for peak performance, but only when there’s high confidence that no leaks are happening. Environment variable: **QUARKUS_DATASOURCE_JDBC_DETECT_STATEMENT_LEAKS**	boolean	**true**
quarkus.datasource.jdbc.new-connection-sql Query executed when first using a connection. Environment variable: **QUARKUS_DATASOURCE_JDBC_NEW_CONNECTION_SQL**	string	
quarkus.datasource.jdbc.validation-query-sql Query executed to validate a connection. Environment variable: **QUARKUS_DATASOURCE_JDBC_VALIDATION_QUERY_SQL**	string	

quarkus.datasource.jdbc.pooling-enabled Disable pooling to prevent reuse of Connections. Use this when an external pool manages the life-cycle of Connections. Environment variable: **QUARKUS_DATASOURCE_JDBC_POOLING_ENABLED**	boolean	**true**
quarkus.datasource.jdbc.transaction-requirement Require an active transaction when acquiring a connection. Recommended for production. WARNING: Some extensions acquire connections without holding a transaction for things like schema updates and schema validation. Setting this setting to STRICT may lead to failures in those cases. Environment variable: **QUARKUS_DATASOURCE_JDBC_TRANSACTION_REQUIREMENT**	**off**, **warn**, **strict**	
quarkus.datasource.jdbc.tracing.enabled Enable JDBC tracing. Environment variable: **QUARKUS_DATASOURCE_JDBC_TRACING_ENABLED**	boolean	**false if quarkus.datasource.jdbc.tracing=false and true if quarkus.datasource.jdbc.tracing=true**
quarkus.datasource.jdbc.tracing.trace-with-active-span-only Trace calls with active Spans only Environment variable: **QUARKUS_DATASOURCE_JDBC_TRACING_TRACE_WITH_ACTIVE_SPAN_ONLY**	boolean	**false**

quarkus.datasource.jdbc.tracing.ignore-for-tracing Ignore specific queries from being traced Environment variable: **QUARKUS_DATASOURCE_JDBC_TRACING_IGNORE_FOR_TRACING**	string	**Ignore specific queries from being traced, multiple queries can be specified separated by semicolon, double quotes should be escaped with \**
quarkus.datasource.jdbc.telemetry.enabled Enable OpenTelemetry JDBC instrumentation. Environment variable: **QUARKUS_DATASOURCE_JDBC_TELEMETRY_ENABLED**	boolean	**false if quarkus.datasource.jdbc.telemetry=false and true if quarkus.datasource.jdbc.telemetry=true**

quarkus.datasource.jdbc.additional-jdbc-properties Other unspecified properties to be passed to the JDBC driver when creating new connections. Environment variable: **QUARKUS_DATASOURCE_JDBC_ADDITIONAL_JDBC_PROPERTIES**	**Map<String,String>**	
Additional named datasources	Type	Default
🔒 **quarkus.datasource."datasource-name".jdbc** If we create a JDBC datasource for this datasource. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC**	boolean	**true**
🔒 **quarkus.datasource."datasource-name".jdbc.driver** The datasource driver class name Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_DRIVER**	string	
🔒 **quarkus.datasource."datasource-name".jdbc.transactions** Whether we want to use regular JDBC transactions, XA, or disable all transactional capabilities. When enabling XA you will need a driver implementing **javax.sql.XADataSource**. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_TRANSACTIONS**	**enabled**, **xa**, **disabled**	**enabled**
🔒 **quarkus.datasource."datasource-name".jdbc.enable-metrics** Enable datasource metrics collection. If unspecified, collecting metrics will be enabled by default if a metrics extension is active. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_ENABLE_METRICS**	boolean	
🔒 **quarkus.datasource."datasource-name".jdbc.tracing** Enable JDBC tracing. Disabled by default. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_TRACING**	boolean	**false**

🔒 **quarkus.datasource."datasource-name".jdbc.telemetry** Enable OpenTelemetry JDBC instrumentation. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_TELEMETRY**	boolean	**false**
quarkus.datasource."datasource-name".jdbc.url The datasource URL Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_URL**	string	
quarkus.datasource."datasource-name".jdbc.initial-size The initial size of the pool. Usually you will want to set the initial size to match at least the minimal size, but this is not enforced so to allow for architectures which prefer a lazy initialization of the connections on boot, while being able to sustain a minimal pool size after boot. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_INITIAL_SIZE**	int	
quarkus.datasource."datasource-name".jdbc.min-size The datasource pool minimum size Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_MIN_SIZE**	int	**0**
quarkus.datasource."datasource-name".jdbc.max-size The datasource pool maximum size Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_MAX_SIZE**	int	**20**
quarkus.datasource."datasource-name".jdbc.background-validation-interval The interval at which we validate idle connections in the background. Set to **0** to disable background validation. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_BACKGROUND_VALIDATION_INTERVAL**	Duration ?	**2M**

quarkus.datasource."datasource-name".jdbc.foreground-validation-interval Perform foreground validation on connections that have been idle for longer than the specified interval. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_FOREGROUND_VALIDATION_INTERVAL**	Duration ?	
quarkus.datasource."datasource-name".jdbc.acquisition-timeout The timeout before cancelling the acquisition of a new connection Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_ACQUISITION_TIMEOUT**	Duration ?	**5**
quarkus.datasource."datasource-name".jdbc.leak-detection-interval The interval at which we check for connection leaks. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_LEAK_DETECTION_INTERVAL**	Duration ?	**This feature is disabled by default.**
quarkus.datasource."datasource-name".jdbc.idle-removal-interval The interval at which we try to remove idle connections. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_IDLE_REMOVAL_INTERVAL**	Duration ?	**5M**
quarkus.datasource."datasource-name".jdbc.max-lifetime The max lifetime of a connection. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_MAX_LIFETIME**	Duration ?	**By default, there is no restriction on the lifespan of a connection.**

quarkus.datasource."datasource-name".jdbc.transaction-isolation-level The transaction isolation level. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_TRANSACTION_ISOLATION_LEVEL**	**undefined**, **none**, **read-uncommitted**, **read-committed**, **repeatable-read**, **serializable**	
quarkus.datasource."datasource-name".jdbc.extended-leak-report Collect and display extra troubleshooting info on leaked connections. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_EXTENDED_LEAK_REPORT**	boolean	**false**
quarkus.datasource."datasource-name".jdbc.flush-on-close Allows connections to be flushed upon return to the pool. It's not enabled by default. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_FLUSH_ON_CLOSE**	boolean	**false**
quarkus.datasource."datasource-name".jdbc.detect-statement-leaks When enabled, Agroal will be able to produce a warning when a connection is returned to the pool without the application having closed all open statements. This is unrelated with tracking of open connections. Disable for peak performance, but only when there's high confidence that no leaks are happening. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_DETECT_STATEMENT_LEAKS**	boolean	**true**
quarkus.datasource."datasource-name".jdbc.new-connection-sql Query executed when first using a connection. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_NEW_CONNECTION_SQL**	string	

quarkus.datasource."datasource-name".jdbc.validation-query-sql Query executed to validate a connection. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_VALIDATION_QUERY_SQL**	string	
quarkus.datasource."datasource-name".jdbc.pooling-enabled Disable pooling to prevent reuse of Connections. Use this when an external pool manages the life-cycle of Connections. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_POOLING_ENABLED**	boolean	**true**
quarkus.datasource."datasource-name".jdbc.transaction-requirement Require an active transaction when acquiring a connection. Recommended for production. WARNING: Some extensions acquire connections without holding a transaction for things like schema updates and schema validation. Setting this setting to STRICT may lead to failures in those cases. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_TRANSACTION_REQUIREMENT**	**off**, **warn**, **strict**	
quarkus.datasource."datasource-name".jdbc.additional-jdbc-properties Other unspecified properties to be passed to the JDBC driver when creating new connections. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_ADDITIONAL_JDBC_PROPERTIES**	**Map<String,String>**	

quarkus.datasource."datasource-name".jdbc.tracing.enabled Enable JDBC tracing. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_TRACING_ENABLED**	boolean	**false if quarkus.datasource.jdbc.tracing=false and true if quarkus.datasource.jdbc.tracing=true**
quarkus.datasource."datasource-name".jdbc.tracing.trace-with-active-span-only Trace calls with active Spans only Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_TRACING_TRACE_WITH_ACTIVE_SPAN_ONLY**	boolean	**false**

quarkus.datasource."datasource-name".jdbc.tracing.ignore-for-tracing Ignore specific queries from being traced Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_TRACING_IGNORE_FOR_TRACING**	string	**Ignore specific queries from being traced, multiple queries can be specified separated by semicolon, double quotes should be escaped with \**
quarkus.datasource."datasource-name".jdbc.telemetry.enabled Enable OpenTelemetry JDBC instrumentation. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__JDBC_TELEMETRY_ENABLED**	boolean	**false if quarkus.datasource.jdbc.telemetry=false and true if quarkus.datasource.jdbc.telemetry=true**

ABOUT THE DURATION FORMAT

To write duration values, use the standard **java.time.Duration** format. See the Duration#parse() javadoc for more information.

You can also use a simplified format, starting with a number:

- If the value is only a number, it represents time in seconds.
- If the value is a number followed by **ms**, it represents time in milliseconds.

In other cases, the simplified format is translated to the **java.time.Duration** format for parsing:

- If the value is a number followed by **h**, **m**, or **s**, it is prefixed with **PT**.
- If the value is a number followed by **d**, it is prefixed with **P**.

1.4.3. JDBC URL reference

Each of the supported databases contains different JDBC URL configuration options. The following section gives an overview of each database URL and a link to the official documentation.

1.4.3.1. DB2

jdbc:db2://<serverName>[:<portNumber>]/<databaseName>[:<key1>=<value>;[<key2>=<value2>;]]

Example

jdbc:db2://localhost:50000/MYDB:user=dbadm;password=dbadm;

For more information on URL syntax and additional supported options, see the official documentation.

1.4.3.2. Derby

jdbc:derby:[//serverName[:portNumber]/][memory:]databaseName[;property=value[;property=value]]

Example

jdbc:derby://localhost:1527/myDB, **jdbc:derby:memory:myDB;create=true**

Derby is an embedded database that can run as a server, based on a file, or can run completely in memory. All of these options are available as listed above.

For more information, see the official documentation.

1.4.3.3. H2

jdbc:h2:{ {.|mem:}[name] | [file:]fileName | {tcp|ssl}:[//]server[:port][,server2[:port]]/name }[;key=value…]

Example

jdbc:h2:tcp://localhost/~/test, **jdbc:h2:mem:myDB**

H2 is a database that can run in embedded or server mode. It can use a file storage or run entirely in memory. All of these options are available as listed above.

For more information, see the official documentation.

1.4.3.4. MariaDB

jdbc:mariadb:[replication:|failover:|sequential:|aurora:]//<hostDescription>[,<hostDescription>…]/[database][?<key1>=<value1>[&<key2>=<value2>]] hostDescription:: **<host>[:<portnumber>] or address=(host=<host>)[(port=<portnumber>)][(type=(master|slave))]**

Example

```
jdbc:mariadb://localhost:3306/test
```

For more information, see the official documentation.

1.4.3.5. Microsoft SQL server

jdbc:sqlserver://[serverName[\instanceName][:portNumber]][;property=value[;property=value]]

Example

```
jdbc:sqlserver://localhost:1433;databaseName=AdventureWorks
```

The Microsoft SQL Server JDBC driver works essentially the same as the others.

For more information, see the official documentation.

1.4.3.6. MySQL

jdbc:mysql:[replication:|failover:|sequential:|aurora:]//<hostDescription>[,<hostDescription>…]/[database][?<key1>=<value1>[&<key2>=<value2>]] hostDescription:: **<host>[:<portnumber>] or address=(host=<host>)[(port=<portnumber>)][(type=(master|slave))]**

Example

```
jdbc:mysql://localhost:3306/test
```

For more information, see the official documentation.

1.4.3.6.1. MySQL limitations

When compiling a Quarkus application to a native image, the MySQL support for JMX and Oracle Cloud Infrastructure (OCI) integrations are disabled as they are incompatible with GraalVM native images.

- The lack of JMX support is a natural consequence of running in native mode and is unlikely to be resolved.
- The integration with OCI is not supported.

1.4.3.7. Oracle

jdbc:oracle:driver_type:@database_specifier

Example

```
jdbc:oracle:thin:@localhost:1521/ORCL_SVC
```

For more information, see the official documentation.

1.4.3.8. PostgreSQL

jdbc:postgresql:[//][host][:port][/database][?key=value…]

Example

jdbc:postgresql://localhost/test

The defaults for the different parts are as follows:

host
: localhost

port
: 5432

database
: same name as the username

For more information about additional parameters, see the official documentation.

1.4.4. Quarkus extensions and database drivers reference

The following tables list the built-in **db-kind** values, the corresponding Quarkus extensions, and the JDBC drivers used by those extensions.

When using one of the built-in datasource kinds, the JDBC and Reactive drivers are resolved automatically to match the values from these tables.

Table 1.1. Database platform kind to JDBC driver mapping

Database kind	Quarkus extension	Drivers
db2	**quarkus-jdbc-db2**	• JDBC: **com.ibm.db2.jcc.DB2Driver** • XA: **com.ibm.db2.jcc.DB2XADataSource**
derby	**quarkus-jdbc-derby**	• JDBC: **org.apache.derby.jdbc.ClientDriver** • XA: **org.apache.derby.jdbc.ClientXADataSource**
h2	**quarkus-jdbc-h2**	• JDBC: **org.h2.Driver** • XA: **org.h2.jdbcx.JdbcDataSource**

Database kind	Quarkus extension	Drivers
mariadb	**quarkus-jdbc-mariadb**	• JDBC: **org.mariadb.jdbc.Driver** • XA: **org.mariadb.jdbc.MySQLDataSource**
mssql	**quarkus-jdbc-mssql**	• JDBC: **com.microsoft.sqlserver.jdbc.SQLServerDriver** • XA: **com.microsoft.sqlserver.jdbc.SQLServerXADataSource**
mysql	**quarkus-jdbc-mysql**	• JDBC: **com.mysql.cj.jdbc.Driver** • XA: **com.mysql.cj.jdbc.MysqlXADataSource**
oracle	**quarkus-jdbc-oracle**	• JDBC: **oracle.jdbc.driver.OracleDriver** • XA: **oracle.jdbc.xa.client.OracleXADataSource**
postgresql	**quarkus-jdbc-postgresql**	• JDBC: **org.postgresql.Driver** • XA: **org.postgresql.xa.PGXADataSource**

Table 1.2. Database kind to Reactive driver mapping

Database kind	Quarkus extension	Driver
oracle	**reactive-oracle-client**	**io.vertx.oracleclient.spi.OracleDriver**
mysql	**reactive-mysql-client**	**io.vertx.mysqlclient.spi.MySQLDriver**
mssql	**reactive-mssql-client**	**io.vertx.mssqlclient.spi.MSSQLDriver**
postgresql	**reactive-pg-client**	**io.vertx.pgclient.spi.PgDriver**
db2	**reactive-db2-client**	**io.vertx.db2client.spi.DB2Driver**

TIP

This automatic resolution is applicable in most cases so that driver configuration is not needed.

1.4.5. Reactive datasource configuration reference

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
🔒 **quarkus.datasource.reactive** If we create a Reactive datasource for this datasource. Environment variable: **QUARKUS_DATASOURCE_REACTIVE**	boolean	**true**
quarkus.datasource.reactive.cache-prepared-statements Whether prepared statements should be cached on the client side. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_CACHE_PREPARED_STATEMENTS**	boolean	**false**
quarkus.datasource.reactive.url The datasource URLs. If multiple values are set, this datasource will create a pool with a list of servers instead of a single server. The pool uses round-robin load balancing for server selection during connection establishment. Note that certain drivers might not accommodate multiple values in this context. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_URL**	list of string	
quarkus.datasource.reactive.max-size The datasource pool maximum size. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_MAX_SIZE**	int	**20**
quarkus.datasource.reactive.event-loop-size When a new connection object is created, the pool assigns it an event loop. When **#event-loop-size** is set to a strictly positive value, the pool assigns as many event loops as specified, in a round-robin fashion. By default, the number of event loops configured or calculated by Quarkus is used. If **#event-loop-size** is set to zero or a negative value, the pool assigns the current event loop to the new connection. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_EVENT_LOOP_SIZE**	int	

quarkus.datasource.reactive.trust-all Whether all server certificates should be trusted. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_TRUST_ALL**	boolean	**false**
quarkus.datasource.reactive.trust-certificate-pem PEM Trust config is disabled by default. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_TRUST_CERTIFICATE_PEM**	boolean	**false**
quarkus.datasource.reactive.trust-certificate-pem.certs Comma-separated list of the trust certificate files (Pem format). Environment variable: **QUARKUS_DATASOURCE_REACTIVE_TRUST_CERTIFICATE_PEM_CERTS**	list of string	
quarkus.datasource.reactive.trust-certificate-jks JKS config is disabled by default. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_TRUST_CERTIFICATE_JKS**	boolean	**false**
quarkus.datasource.reactive.trust-certificate-jks.path Path of the key file (JKS format). Environment variable: **QUARKUS_DATASOURCE_REACTIVE_TRUST_CERTIFICATE_JKS_PATH**	string	
quarkus.datasource.reactive.trust-certificate-jks.password Password of the key file. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_TRUST_CERTIFICATE_JKS_PASSWORD**	string	
quarkus.datasource.reactive.trust-certificate-pfx PFX config is disabled by default. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_TRUST_CERTIFICATE_PFX**	boolean	**false**

quarkus.datasource.reactive.trust-certificate-pfx.path Path to the key file (PFX format). Environment variable: **QUARKUS_DATASOURCE_REACTIVE_TRUST_CERTIFICATE_PFX_PATH**	string	
quarkus.datasource.reactive.trust-certificate-pfx.password Password of the key. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_TRUST_CERTIFICATE_PFX_PASSWORD**	string	
quarkus.datasource.reactive.key-certificate-pem PEM Key/cert config is disabled by default. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_KEY_CERTIFICATE_PEM**	boolean	**false**
quarkus.datasource.reactive.key-certificate-pem.keys Comma-separated list of the path to the key files (Pem format). Environment variable: **QUARKUS_DATASOURCE_REACTIVE_KEY_CERTIFICATE_PEM_KEYS**	list of string	
quarkus.datasource.reactive.key-certificate-pem.certs Comma-separated list of the path to the certificate files (Pem format). Environment variable: **QUARKUS_DATASOURCE_REACTIVE_KEY_CERTIFICATE_PEM_CERTS**	list of string	
quarkus.datasource.reactive.key-certificate-jks JKS config is disabled by default. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_KEY_CERTIFICATE_JKS**	boolean	**false**
quarkus.datasource.reactive.key-certificate-jks.path Path of the key file (JKS format). Environment variable: **QUARKUS_DATASOURCE_REACTIVE_KEY_CERTIFICATE_JKS_PATH**	string	

quarkus.datasource.reactive.key-certificate-jks.password Password of the key file. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_KEY_CERTIFICATE_JKS_PASSWORD**	string	
quarkus.datasource.reactive.key-certificate-pfx PFX config is disabled by default. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_KEY_CERTIFICATE_PFX**	boolean	**false**
quarkus.datasource.reactive.key-certificate-pfx.path Path to the key file (PFX format). Environment variable: **QUARKUS_DATASOURCE_REACTIVE_KEY_CERTIFICATE_PFX_PATH**	string	
quarkus.datasource.reactive.key-certificate-pfx.password Password of the key. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_KEY_CERTIFICATE_PFX_PASSWORD**	string	
quarkus.datasource.reactive.reconnect-attempts The number of reconnection attempts when a pooled connection cannot be established on first try. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_RECONNECT_ATTEMPTS**	int	**0**
quarkus.datasource.reactive.reconnect-interval The interval between reconnection attempts when a pooled connection cannot be established on first try. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_RECONNECT_INTERVAL**	Duration ?	**PT1S**
quarkus.datasource.reactive.hostname-verification-algorithm The hostname verification algorithm to use in case the server’s identity should be checked. Should be HTTPS, LDAPS or an empty string. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_HOSTNAME_VERIFICATION_ALGORITHM**	string	

quarkus.datasource.reactive.idle-timeout The maximum time a connection remains unused in the pool before it is closed. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_IDLE_TIMEOUT**	Duration ?	**no timeout**
quarkus.datasource.reactive.shared Set to true to share the pool among datasources. There can be multiple shared pools distinguished by name, when no specific name is set, the **__vertx.DEFAULT** name is used. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_SHARED**	boolean	**false**
quarkus.datasource.reactive.name Set the pool name, used when the pool is shared among datasources, otherwise ignored. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_NAME**	string	
quarkus.datasource.reactive.additional-properties Other unspecified properties to be passed through the Reactive SQL Client directly to the database when new connections are initiated. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_ADDITIONAL_PROPERTIES**	**Map<String,String>**	
Additional named datasources	Type	Default
🔒 **quarkus.datasource."datasource-name".reactive** If we create a Reactive datasource for this datasource. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE**	boolean	**true**
quarkus.datasource."datasource-name".reactive.cache-prepared-statements Whether prepared statements should be cached on the client side. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_CACHE_PREPARED_STATEMENTS**	boolean	**false**

quarkus.datasource."datasource-name".reactive.url The datasource URLs. If multiple values are set, this datasource will create a pool with a list of servers instead of a single server. The pool uses round-robin load balancing for server selection during connection establishment. Note that certain drivers might not accommodate multiple values in this context. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_URL**	list of string	
quarkus.datasource."datasource-name".reactive.max-size The datasource pool maximum size. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_MAX_SIZE**	int	**20**
quarkus.datasource."datasource-name".reactive.event-loop-size When a new connection object is created, the pool assigns it an event loop. When **#event-loop-size** is set to a strictly positive value, the pool assigns as many event loops as specified, in a round-robin fashion. By default, the number of event loops configured or calculated by Quarkus is used. If **#event-loop-size** is set to zero or a negative value, the pool assigns the current event loop to the new connection. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_EVENT_LOOP_SIZE**	int	
quarkus.datasource."datasource-name".reactive.trust-all Whether all server certificates should be trusted. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_TRUST_ALL**	boolean	**false**
quarkus.datasource."datasource-name".reactive.trust-certificate-pem PEM Trust config is disabled by default. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_TRUST_CERTIFICATE_PEM**	boolean	**false**

quarkus.datasource."datasource-name".reactive.trust-certificate-pem.certs Comma-separated list of the trust certificate files (Pem format). Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_TRUST_CERTIFICATE_PEM_CERTS**	list of string	
quarkus.datasource."datasource-name".reactive.trust-certificate-jks JKS config is disabled by default. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_TRUST_CERTIFICATE_JKS**	boolean	**false**
quarkus.datasource."datasource-name".reactive.trust-certificate-jks.path Path of the key file (JKS format). Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_TRUST_CERTIFICATE_JKS_PATH**	string	
quarkus.datasource."datasource-name".reactive.trust-certificate-jks.password Password of the key file. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_TRUST_CERTIFICATE_JKS_PASSWORD**	string	
quarkus.datasource."datasource-name".reactive.trust-certificate-pfx PFX config is disabled by default. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_TRUST_CERTIFICATE_PFX**	boolean	**false**
quarkus.datasource."datasource-name".reactive.trust-certificate-pfx.path Path to the key file (PFX format). Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_TRUST_CERTIFICATE_PFX_PATH**	string	

quarkus.datasource."datasource-name".reactive.trust-certificate-pfx.password Password of the key. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_TRUST_CERTIFICATE_PFX_PASSWORD**	string	
quarkus.datasource."datasource-name".reactive.key-certificate-pem PEM Key/cert config is disabled by default. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_KEY_CERTIFICATE_PEM**	boolean	**false**
quarkus.datasource."datasource-name".reactive.key-certificate-pem.keys Comma-separated list of the path to the key files (Pem format). Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_KEY_CERTIFICATE_PEM_KEYS**	list of string	
quarkus.datasource."datasource-name".reactive.key-certificate-pem.certs Comma-separated list of the path to the certificate files (Pem format). Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_KEY_CERTIFICATE_PEM_CERTS**	list of string	
quarkus.datasource."datasource-name".reactive.key-certificate-jks JKS config is disabled by default. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_KEY_CERTIFICATE_JKS**	boolean	**false**
quarkus.datasource."datasource-name".reactive.key-certificate-jks.path Path of the key file (JKS format). Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_KEY_CERTIFICATE_JKS_PATH**	string	

quarkus.datasource."datasource-name".reactive.key-certificate-jks.password Password of the key file. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_KEY_CERTIFICATE_JKS_PASSWORD**	string	
quarkus.datasource."datasource-name".reactive.key-certificate-pfx PFX config is disabled by default. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_KEY_CERTIFICATE_PFX**	boolean	**false**
quarkus.datasource."datasource-name".reactive.key-certificate-pfx.path Path to the key file (PFX format). Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_KEY_CERTIFICATE_PFX_PATH**	string	
quarkus.datasource."datasource-name".reactive.key-certificate-pfx.password Password of the key. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_KEY_CERTIFICATE_PFX_PASSWORD**	string	
quarkus.datasource."datasource-name".reactive.reconnect-attempts The number of reconnection attempts when a pooled connection cannot be established on first try. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_RECONNECT_ATTEMPTS**	int	**0**
quarkus.datasource."datasource-name".reactive.reconnect-interval The interval between reconnection attempts when a pooled connection cannot be established on first try. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_RECONNECT_INTERVAL**	Duration ?	**PT1S**

quarkus.datasource."datasource-name".reactive.hostname-verification-algorithm The hostname verification algorithm to use in case the server's identity should be checked. Should be HTTPS, LDAPS or an empty string. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_HOSTNAME_VERIFICATION_ALGORITHM**	string	
quarkus.datasource."datasource-name".reactive.idle-timeout The maximum time a connection remains unused in the pool before it is closed. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_IDLE_TIMEOUT**	Duration ?	**no timeout**
quarkus.datasource."datasource-name".reactive.shared Set to true to share the pool among datasources. There can be multiple shared pools distinguished by name, when no specific name is set, the **__vertx.DEFAULT** name is used. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_SHARED**	boolean	**false**
quarkus.datasource."datasource-name".reactive.name Set the pool name, used when the pool is shared among datasources, otherwise ignored. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_NAME**	string	
quarkus.datasource."datasource-name".reactive.additional-properties Other unspecified properties to be passed through the Reactive SQL Client directly to the database when new connections are initiated. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_ADDITIONAL_PROPERTIES**	**Map<String,String>**	

ABOUT THE DURATION FORMAT

To write duration values, use the standard **java.time.Duration** format. See the Duration#parse() javadoc for more information.

You can also use a simplified format, starting with a number:

- If the value is only a number, it represents time in seconds.
- If the value is a number followed by **ms**, it represents time in milliseconds.

In other cases, the simplified format is translated to the **java.time.Duration** format for parsing:

- If the value is a number followed by **h**, **m**, or **s**, it is prefixed with **PT**.
- If the value is a number followed by **d**, it is prefixed with **P**.

1.4.5.1. Reactive DB2 configuration

Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
quarkus.datasource.reactive.db2.ssl Whether SSL/TLS is enabled. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_DB2_SSL**	boolean	**false**
Additional named datasources	Type	Default
quarkus.datasource."datasource-name".reactive.db2.ssl Whether SSL/TLS is enabled. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_DB2_SSL**	boolean	**false**

1.4.5.2. Reactive MariaDB/MySQL specific configuration

Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default

quarkus.datasource.reactive.mysql.charset Charset for connections. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_MYSQL_CHARSET**	string	
quarkus.datasource.reactive.mysql.collation Collation for connections. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_MYSQL_COLLATION**	string	
quarkus.datasource.reactive.mysql.ssl-mode Desired security state of the connection to the server. See MySQL Reference Manual. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_MYSQL_SSL_MODE**	**disabled**, **preferred**, **required**, **verify-ca**, **verify-identity**	**disabled**
quarkus.datasource.reactive.mysql.connection-timeout Connection timeout in seconds Environment variable: **QUARKUS_DATASOURCE_REACTIVE_MYSQL_CONNECTION_TIMEOUT**	int	
quarkus.datasource.reactive.mysql.authentication-plugin The authentication plugin the client should use. By default, it uses the plugin name specified by the server in the initial handshake packet. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_MYSQL_AUTHENTICATION_PLUGIN**	**default**, **mysql-clear-password**, **mysql-native-password**, **sha256-password**, **caching-sha2-password**	**default**

quarkus.datasource.reactive.mysql.pipelining-limit The maximum number of inflight database commands that can be pipelined. By default, pipelining is disabled. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_MYSQL_PIPELINING_LIMIT**	int	
quarkus.datasource.reactive.mysql.use-affected-rows Whether to return the number of rows matched by the *WHERE* clause in *UPDATE* statements, instead of the number of rows actually changed. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_MYSQL_USE_AFFECTED_ROWS**	boolean	**false**
Additional named datasources	Type	Default
quarkus.datasource."datasource-name".reactive.mysql.charset Charset for connections. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_MYSQL_CHARSET**	string	
quarkus.datasource."datasource-name".reactive.mysql.collation Collation for connections. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_MYSQL_COLLATION**	string	
quarkus.datasource."datasource-name".reactive.mysql.ssl-mode Desired security state of the connection to the server. See MySQL Reference Manual. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_MYSQL_SSL_MODE**	**disabled**, **preferred**, **required**, **verify-ca**, **verify-identity**	**disabled**

quarkus.datasource."datasource-name".reactive.mysql.connection-timeout Connection timeout in seconds Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_MYSQL_CONNECTION_TIMEOUT**	int	
quarkus.datasource."datasource-name".reactive.mysql.authentication-plugin The authentication plugin the client should use. By default, it uses the plugin name specified by the server in the initial handshake packet. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_MYSQL_AUTHENTICATION_PLUGIN**	**default**, **mysql-clear-password**, **mysql-native-password**, **sha256-password**, **caching-sha2-password**	**default**
quarkus.datasource."datasource-name".reactive.mysql.pipelining-limit The maximum number of inflight database commands that can be pipelined. By default, pipelining is disabled. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_MYSQL_PIPELINING_LIMIT**	int	
quarkus.datasource."datasource-name".reactive.mysql.use-affected-rows Whether to return the number of rows matched by the *WHERE* clause in *UPDATE* statements, instead of the number of rows actually changed. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_MYSQL_USE_AFFECTED_ROWS**	boolean	**false**

1.4.5.3. Reactive Microsoft SQL server-specific configuration

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
quarkus.datasource.reactive.mssql.packet-size The desired size (in bytes) for TDS packets. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_MSSQL_PACKET_SIZE**	int	
quarkus.datasource.reactive.mssql.ssl Whether SSL/TLS is enabled. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_MSSQL_SSL**	boolean	**false**
Additional named datasources	Type	Default
quarkus.datasource."datasource-name".reactive.mssql.packet-size The desired size (in bytes) for TDS packets. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_MSSQL_PACKET_SIZE**	int	
quarkus.datasource."datasource-name".reactive.mssql.ssl Whether SSL/TLS is enabled. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_MSSQL_SSL**	boolean	**false**

1.4.5.4. Reactive Oracle-specific configuration

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Additional named datasources	Type	Default

1.4.5.5. Reactive PostgreSQL-specific configuration

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default

quarkus.datasource.reactive.postgresql.pipelining-limit The maximum number of inflight database commands that can be pipelined. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_POSTGRESQL_PIPELINING_LIMIT**	int	
quarkus.datasource.reactive.postgresql.ssl-mode SSL operating mode of the client. See Protection Provided in Different Modes. Environment variable: **QUARKUS_DATASOURCE_REACTIVE_POSTGRESQL_SSL_MODE**	**disable**, **allow**, **prefer**, **require**, **verify-ca**, **verify-full**	**disable**
Additional named datasources	Type	Default
quarkus.datasource."datasource-name".reactive.postgresql.pipelining-limit The maximum number of inflight database commands that can be pipelined. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_POSTGRESQL_PIPELINING_LIMIT**	int	
quarkus.datasource."datasource-name".reactive.postgresql.ssl-mode SSL operating mode of the client. See Protection Provided in Different Modes. Environment variable: **QUARKUS_DATASOURCE__DATASOURCE_NAME__REACTIVE_POSTGRESQL_SSL_MODE**	**disable**, **allow**, **prefer**, **require**, **verify-ca**, **verify-full**	**disable**

1.4.6. Reactive datasource URL reference

1.4.6.1. DB2

db2://[user[:[password]]@]host[:port][/database][?<key1>=<value1>[&<key2>=<value2>]]

Example

```
db2://dbuser:secretpassword@database.server.com:50000/mydb
```

Currently, the client supports the following parameter keys:

- **host**

- **port**
- **user**
- **password**
- **database**

NOTE

Configuring parameters in the connection URL overrides the default properties.

1.4.6.2. Microsoft SQL server

sqlserver://[user[:[password]]@]host[:port][/database][?<key1>=<value1>[&<key2>=<value2>]]

Example

```
sqlserver://dbuser:secretpassword@database.server.com:1433/mydb
```

Currently, the client supports the following parameter keys:

- **host**
- **port**
- **user**
- **password**
- **database**

NOTE

Configuring parameters in the connection URL overrides the default properties.

1.4.6.3. MySQL / MariaDB

mysql://[user[:[password]]@]host[:port][/database][?<key1>=<value1>[&<key2>=<value2>]]

Example

```
mysql://dbuser:secretpassword@database.server.com:3211/mydb
```

Currently, the client supports the following parameter keys (case-insensitive):

- **host**
- **port**
- **user**
- **password**
- **schema**
- **socket**

- **useAffectedRows**

> NOTE
>
> Configuring parameters in the connection URL overrides the default properties.

1.4.6.4. Oracle

1.4.6.4.1. EZConnect format

oracle:thin:@[[protocol:]//]host[:port][/service_name][:server_mode][/instance_name][?connection properties]

Example

```
oracle:thin:@mydbhost1:5521/mydbservice?connect_timeout=10sec
```

1.4.6.4.2. TNS alias format

oracle:thin:@<alias_name>[?connection properties]

Example

```
oracle:thin:@prod_db?TNS_ADMIN=/work/tns/
```

1.4.6.5. PostgreSQL

postgresql://[user[:[password]]@]host[:port][/database][?<key1>=<value1>[&<key2>=<value2>]]

Example

```
postgresql://dbuser:secretpassword@database.server.com:5432/mydb
```

Currently, the client supports:

- Following parameter keys:
  - **host**
  - **port**
  - **user**
  - **password**
  - **dbname**
  - **sslmode**
- Additional properties, such as:
  - **application_name**
  - **fallback_application_name**
  - **search_path**
  - **options**

NOTE

Configuring parameters in the connection URL overrides the default properties.